A. ***(17 marks)***

Fill in the blanks in the statements below with the word or phrase wh

1. One main "stream" in the evolution of UNIX is the BSD stream
   System III / System V stream.

2. An operating system can be organized according to "layers of abstraction", where each layer defines a
   virtual machine .

3. A kernel is that portion of the operating system which provides only the minimum of services necessary for implementing additional O/S services

4. The acronym PPID stands for parent process ID .

5 A UNIX process is always in a well-defined state. One state that a process could be in is the
   any one of SZOMB, SIDL, SSTOP, SRUN, SSLEEP state.

6. In UNIX, the operation of process "spawning" is split into two separate operations,
   fork and exec .

7. The memory areas in a UNIX process include text, data, bss, and any one of {u-dot, stack, heap, etc.}

8. The shell (whether */bin/sh* or */bin/csh*) is both a command interpreter and a programming language.

9. The name of the *csh* variable which defines the list of paths searched for commands is
   path MUST be lowercase

10. In *sh*, the built-in command export is used to tell the shell that a particular (user-created) shell variable is to be treated as an environment variable.

11. A signal is a report to a process that a (an) asynchronous event has occurred.

12. The default disposition for a process on receipt of a {SIGABRT, SIGQUIT, others} signal is abnormal termination.

13. The operation of grafting one file system onto another is called, in UNIX terminology, mounting a file system.

14. The file name "." (i.e. "dot") refers to the current working directory.

15. A program with which you can interactively debug C programs on a NetBSD UNIX system is {gdb, ddd}.

B. ***(1+2+1+1+1+1 = 7 marks)***

Answer each of the following questions with a very short answer.

1. Give the name of a kernel call that allows a process to determine why (or how) another process terminated.

   wait()

2. What is the filename wildcard pattern that will match any file name ending in either ".z" or ".c".

   *.zc

3. What is the shell constuct used to invoke a command in a subshell? Give that construct.

   ()

4. In the UNIX file system, how many inodes can a file have?

   1

5. Suppose a disk with physical block size of 512 bytes has a UNIX file system stored on it. At what physical block number will the superblock (of that file system) start?

   16

6. A file is has UID=91, GID=23, and permissions 0764 (764 octal). A process with UID=215 and GID=23 tries to open the file for write access. Will the open operation be permitted?

   yes

   To help you answer the question above, here is some relevant content from <ufs/ufs/dinode.h>:

```
/* File permissions. */
#define IEXEC           0000100         /* Executable. */
#define IWRITE          0000200         /* Writeable. */
#define IREAD           0000400         /* Readable. */
#define ISVTX           0001000         /* Sticky bit. */
#define ISGID           0002000         /* Set-gid. */
#define ISUID           0004000         /* Set-uid. */
```

C. *(6 marks)*

A user wants to concatenate the value of (*sh*) variable `filename`, the string ".", and the value of (*sh*) variable `suffix` and set the value of the (*sh*) variable `newstring` to the result. E.g. if variables `filename` and `suffix` have values as shown

```
$echo $filename
foobar
$echo $suffix
c
```

the user would like to set a user-created shell variable, `newstring`, to the value "`foobar.c`".

Below are 5 different shell commands that the user could type to try to achieve this. Only one of them will achieve the result that the user wants. Which one is it? (Give the number.)

5

Further, for each command below, specify the value of `newstring` after each command is executed. Or if the command will result in an error, say so. Recall that the -n option to *echo(1)* suppresses the output of a trailing newline character.

1. `newstring=echo -n $filename . $suffix`

   error

2. `newstring="echo -n $filename . $suffix"`

   echo -n foobar.c

3. `newstring='echo -n $filename . $suffix'`

   echo -n $filename . $suffix

4. ``newstring=``echo -n $filename`.`echo $suffix``

   `echo -n $filename` . `echo -n $suffix`

5. ``newstring="`echo -n $filename`.`echo $suffix`"``

   foobar.c

D. *(4 marks)*

Consider the following program. Its functionality is to accept, on the command line, any number of pathnames (of files). For each file pathname, the program then obtains inode information for the file (via *stat(2)*), and prints, on the standard output, the name of the file followed by a string which identifies the type of that file. (The program is a simple modification of an example in the Stevens text.

```
#include        <sys/types.h>
#include        <sys/stat.h>
#include        <stdio.h>

int main(int argc, char *argv[])
{
   int   i;
   struct stat buf;
   char  *ptr;

   for (i = 1; i < argc; i++) {
      printf("%s: ", argv[i]);
      if (lstat(argv[i], &buf) < 0) {
```

```
            perror("lstat error");
            continue;
        }

        if (S_ISREG(buf.st_mode))  ptr = "regular";
        else if (S_ISDIR(buf.st_mode))  ptr = "directory";
        else if (S_ISCHR(buf.st_mode))  ptr = "character special";
        else if (S_ISBLK(buf.st_mode))  ptr = "block special";
        else if (S_ISFIFO(buf.st_mode)) ptr = "fifo";
        else if (S_ISLNK(buf.st_mode))  ptr = "symbolic link";
        else                            ptr = "** unknown mode **";
        printf("%s\n", ptr);
    }

    exit(0);
}
```

The program could be improved stylistically by using a switch-statement instead of the extended if-statement for checking the st_mode field (determining the file type) and setting variable ptr to the appropriate string. In the space below give a portion which does this; i.e. in the space below give a portion of code equivalent to

```
        if (S_ISREG(buf.st_mode))  ptr = "regular";
        else if (S_ISDIR(buf.st_mode))  ptr = "directory";
        else if (S_ISCHR(buf.st_mode))  ptr = "character special";
        else if (S_ISBLK(buf.st_mode))  ptr = "block special";
        else if (S_ISFIFO(buf.st_mode)) ptr = "fifo";
        else if (S_ISLNK(buf.st_mode))  ptr = "symbolic link";
        else                            ptr = "** unknown mode **";
```

but which uses a switch-statement and no if-statements. To be able to perform this task, the following excepts from <sys/stat.h> will be helpful:

```
struct stat {
        dev_t     st_dev;               /* inode's device */
        ino_t     st_ino;               /* inode's number */
        mode_t    st_mode;              /* inode type and protection mode */
        nlink_t   st_nlink;             /* number of hard links */
        uid_t     st_uid;               /* user ID of the file's owner */
        gid_t     st_gid;               /* group ID of the file's group */
        dev_t     st_rdev;              /* device type */
        struct    timespec st_atimespec;/* time of last access */
        struct    timespec st_mtimespec;/* time of last data modification */
        struct    timespec st_ctimespec;/* time of last file status change */
        off_t     st_size;              /* file size, in bytes */
        blkcnt_t  st_blocks;            /* blocks allocated for file */
        blksize_t st_blksize;           /* optimal blocksize for I/O */
        u_int32_t st_flags;             /* user defined flags for file */
        u_int32_t st_gen;               /* file generation number */
        int64_t   st_qspare[2];
};

#define _S_IFMT   0170000               /* type of file mask */
#define _S_IFIFO  0010000               /* named pipe (fifo) */
#define _S_IFCHR  0020000               /* character special */
#define _S_IFDIR  0040000               /* directory */
#define _S_IFBLK  0060000               /* block special */
#define _S_IFREG  0100000               /* regular */
```

```
#define _S_IFLNK  0120000              /* symbolic link */

#define S_IFMT   _S_IFMT
#define S_IFIFO  _S_IFIFO
#define S_IFCHR  _S_IFCHR
#define S_IFDIR  _S_IFDIR
#define S_IFBLK  _S_IFBLK
#define S_IFREG  _S_IFREG
#define S_IFLNK  _S_IFLNK

#define S_ISDIR(m)      ((m & _S_IFMT) == _S_IFDIR)     /* directory */
#define S_ISCHR(m)      ((m & _S_IFMT) == _S_IFCHR)     /* char special */
#define S_ISBLK(m)      ((m & _S_IFMT) == _S_IFBLK)     /* block special */
#define S_ISREG(m)      ((m & _S_IFMT) == _S_IFREG)     /* regular file */
#define S_ISFIFO(m)     ((m & _S_IFMT) == _S_IFIFO)     /* fifo */
#define S_ISLNK(m)      ((m & _S_IFMT) == _S_IFLNK)     /* symbolic link */
```

Also recall that the general form of a switch-statement in C is

```
switch (expression) {
    case item1:
        statement1;
        break;
    case item2:
        statement2;
        break;
    .
    .
    .
    case itemn:
        statementn;
        break;
    default:
        statement;
        break;
}
```

Give your code here:

```
switch( buf.st_mode & S_IFMT ){
    case S_IFREG :  ptr = "regular"; break;
    case S_IFDIR :  ptr = "directory"; break;
    case S_IFCHR :  ptr = "character special"; break;
    case S_IFBLK :  ptr = "block special"; break;
    case S_IFFIFO:  ptr = "fifo";
    case S_IFLINK:  ptr = "symbolic link";
    default :  ptr = "** unknown mode **"; break;
}
```

common error: having a construction like

case S_ISCHR(buf.st_mode)

E. (3+3 = 6 marks)

Answer each of the following questions with a short, but precise descriptive answer.

1. What is the program *tee(1)* used for? Give a practical example of its use. I.e. give a realistic command where it is effectively used. If the interpretation of your example command is not obvious, provide an explanation.

tee(1) is used to store into a (permanent) file the bytes passing through a pipe involving tee. I.e. tee(1) reads from stdin, and writes the bytes to the named file as well as to stdout.

e.g. gunzip saveset.tar.gz | tee saveset.tar | tar tvf -

this way the results of the uncompression are stored in saveset.tar and do not need to be recomputed

3. Suppose that you suspect that two files in your directory are one in the same file; i.e. not just a copy, but that there are two pathnames leading to exactly the same file. How would you determine this? Be as specific as possible with respect to what command(s) you would use, and what you would look for.

perform an ls -i on both files. If the reported inode numbers are the same, the two pathnames lead to the same file.

common error: saying to use ls -l and observe the link count

F. (4 marks)

In the space below, write a shell script (in either *sh* or *csh*) that displays its command line arguments, one per line, on the standard output and then exits with success.

```
#! /bin/sh
while [ $# -gt 0 ]
do
        echo $1
        shift
done
```

G. (3+3 = 6 marks)

Answer each of the following questions with a focused discussion-oriented answer. You may wish to use examples and/or diagrams to illustrate points in your answer.

1. There is a *csh* environment variable called *noclobber*. *Csh* tries to use the setting of this variable to prevent accidentally overwriting the existing content of a file when output is redirected. That is, if a user types

```
ls > myfile
```

any existing content of `myfile` would normally (if *noclobber* was not set) be over-written by the output of the *ls* command. However, if *noclobber* is set, and if `myfile` already exists, *csh* will not redirect output to the file. With *noclobber* set, output is written to `myfile` only if `myfile` does not already exist (and then it is created and written to).

Unfortunately, the setting of the noclobber shell variable does not protect a user from overwriting an existing file when using *mv* or *cp*. Why is this? Why can the setting of the *noclobber* variable not protect a user from overwriting an existing file when using *mv* or *cp*?

The noclobber variable is local to the shell. When a mv or cp command is executed (via the shell), first a fork() is performed and then an exec. The exec means that the memory image that was in the newly created child (which was a copy of the parent shell's memory image, including the noclobber setting) is replaced. This means that any information in the memory image - such as the setting of program variables - is wiped out and cannot be known by a subsequent program such as cp or mv.

2. What is the difference between

```
dup(2);
dup(4);
```

and

```
dup2(2,4);
```

What is the difference in the effect of the two code portions?

dup(2); dup(4); duplicates file descriptors 2 and 4 into the next available file descriptors

dup2(2,4); file descriptor 4 is closed. Then file descriptor 2 is duplicated as file descriptor 4.

Thus the first portion of code produces two new file descriptors (duplicates of file descriptors 2 and 4) while the second leaves file descriptors 2 and 4 refering to the same open instance of a file.